The report committee for Anurag Kumar
Certifies that this is the approved version of the following report:

Placement for Structured ASICs

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____
David Z. Pan

_____
Mark McDermott

Placement for Structured ASICs

by

Anurag Kumar, B.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2009

Dedicated to my parents Mrs. Anita Roy and Mr. Kishor Kumar Roy for their love, guidance, and blessing.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude, respect and regards to my advisor Prof. David Z. Pan who guided my research and learning at UT Austin and introduced me to the problem addressed in this report. Special thanks goes to Prof. Mark McDermott for agreeing to be the reader of this report and for introducing me to the basics of circuit optimization techniques during his course VLSI-2. Special thanks goes to eASIC corporation for organizing the placement contest and releasing the benchmarks. Thanks also to Mr. Ashutosh Chakraborty for very interesting discussions regarding placement techniques and implementation of the whole placer. Last but not the least, many thanks to all my colleagues at UTDA lab at UT Austin who gave suggestions and helped me develop this work.

Placement for Structured ASICs

Anurag Kumar, M.S.E.
The University of Texas at Austin, 2009

Supervisor: David Z. Pan

Structured ASICs provide an exciting middle-ground between ASIC and FPGA design styles because they provide trade-off between the high performance of ASIC design and low costs of FPGA design. To fully utilize the benefits of structured ASIC, placement stage must be aware of the modularity of the structured ASIC architecture. This work describes a novel solution to placement of structured ASICs. Integer linear programming formulation is proposed for satisfying the constraints associated with structured ASIC clock architecture. Regularity of the platform is exploited during legalization and wirelength recovery stages to speed-up the detailed placement stage. Our methods show overall wirelength reduction up to 33% and up to 3X speedup compared to other placers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increasing costs and variablity accociated with an ASIC design flow on one hand and unacceptable power and delay penalty associated with FPGA design on the other have forced the semiconductor companies to look for alternative technologies. One viable alternative that has emerged recently is the use of *structured* ASICs. Structured ASICs provide an exciting middle-ground between high performance of ASIC designs and short time-to-market FPGA designs. Structured ASICs use the fact that not all mask-layers provide the same value for the customers. Mask layers that are less useful can be prefabricated thus amortizing their cost over multiple designs [21]. Structured ASIC flow is much simpler than that of traditional ASICs such as signal integrity, power grid optimization are already taken care of by the structured ASIC vendor. Moreover, structured ASICs can be used to implement designs consisting of millions of gates in contrast to FPGAs which can implement a much lesser number of gates. Different structured ASIC vendors provide different architectures but all of them have a fundamental repeated logic element called *tile* which may contain pre-defined combinational logic, small RAM, and registers [3].

Tools for high quality placement and routing need to be developed to fully utilize the benefits of structure ASICs. Placement for structured ASIC requires cells to be mapped exactly on a compatible site, similar to the case for an FPGA. Algorithms used for FPGA placement don't scale well for structured ASIC placement problems because structured ASIC placement may be orders of magnitude larger than FPGA problems [21]. Not only does the placer have to handle millions of cells along with the site compatibility requirement, the task is made much more difficult due to the clocking schemes in structured ASICs. The clocks are built with pre-allocated resources to provide low skew clocking and simplified design flow. This restricts the number of clocked elements which can be placed in proximity of each other. Clocked elements need to be placed in a manner that clock skew and power of clock-tree remains within limit. This work presents an open source placement tool for structured ASICs which can deal with the above mentioned legality and clock constraints.

This report is organized as follows: We describe the intricacies of placement for structured ASIC in Chapter 2. Chapter 3 discusses the previous work done and contributions of this work. Complete placement flow is presented in Chapter 4. Experimental results are presented in Chapter 5. Chapter 6 concludes our paper.

Chapter 2

Problem Description

As discussed earlier, large problem size, site compatible requirements, whitespace requirement for routing tracks and strict clock constraints make the task of placement in structured ASIC very difficult. This work describes the solution to the placement problem of structured ASICs. We base our work on the architecture of the popular Nextreme line of structured ASICs from eASIC Corporation [1]. Nextreme is the most *structured* of structured ASIC solutions with only one via layer available for customization.

There are four types of cells in the Nextreme line of structured ASICs. They are SRAM programmable Logic cells (LCELLS), flip flops (DFF), registers (REG) and memories (BRAM). In the rest of the paper, we will refer to these types of cells with their short names and all these types will collectively referred to as *cells*. The placement problem requires that a cell can only be placed on a location which is reserved for that type of cell. In the basic repeating *tile* of Nextreme architecture the space reserved for LCELL, DFF, REG and BRAM are as shown in Figure 2.1. There are 36 LCELL and 24 DFF columns in each tile. Each of these column accommodate 64 LCELLs and DFFs respectively. A total of 4 REG and 1 BRAM can be accommodated

in each tile. This tile repeats all over the chip with some horizontal and vertical inter-tile whitespace between adjacent tiles. Depending on the size of the netlist to be placed, the number of times these tiles need to be repeated can be configured. Table 5.2 shows the details of two such configured platforms along with the maximum cells of each type that can be accommodated in them.

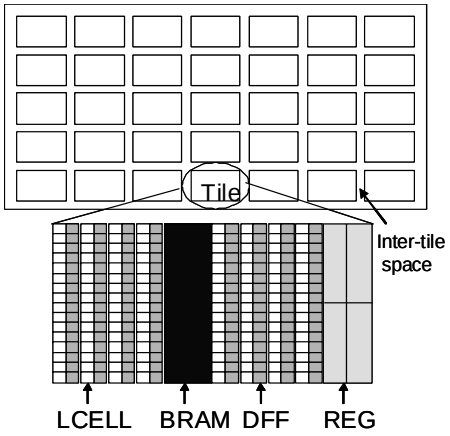

Figure 2.1: Structured ASIC platform.

The sizes of each LCELL, DFF, REG and BRAM in the mapped design netlist are 1x1, 1x1, 8x32 and 16x64. These cells are to be placed according to the reserved column locations described above. The presence of these four different types of cells require that the global placement ensure that the density requirement for each type of cells is met. The presence of whitespaces and interleaving of space for each of the cell types makes the physical location available for each type of cell very discrete.

Because of architectural constraints, only a certain number of different

clocks can go inside a tile. For example, in Figure 2.1, each tile can have $N(=4$ in our benchmarks) different clocks. Moreover, all the cells in one column of DFF can have only one clock type. The clock constraint on placement has big impact on final placement because clock violations may require that the violating cells be reallocated to another region far away from their optimal location.

Chapter 3

Previous Works and Contributions

There are several commercial and academic placers for ASIC designs e.g. [20][9][22][16][4]. These placers implement sophisticated mathematical or min-cut formulations to generate very good quality results for ASIC designs. However, due to site compatibility and hard clock constraints they cannot be directly used for placement of structured ASIC designs. Nevertheless, if the key algorithms in these placers can be somehow used to solve the structured ASIC placement problem, the solutions would be of good quality [7]. Another class of placement tools which are designed for FPGAs can take care of site constraints. However, due to smaller sizes of FPGA based designs, the existing FPGA placement approaches such as [6][11] rely on slow algorithms like simulated annealing which cannot scale to problem sizes frequently encountered in structured ASIC designs.

There is limited research in the domain of placement for structured ASICs. The work in [15] only addresses incremental placement issues in structured ASICs. Companies are currently using existing row-based placers with product specific legalizers or heuristic [17]. Recently, there has been some work for parallel global placement [10] for ASICs. Some work has been done

for placement for FPGA [8]. However, there is a dearth of tools which can handle the clock constraints or exploit the modularity of structured ASIC platforms.

The main technical contributions in this work are:

- An integer linear program (ILP) formulation is proposed to satisfy clock constraints on the type of clocks that can appear in each column of a tile.
- A detail placement flow is used specifically for tile based structured ASIC architectures.
- Regularity of structured ASIC is exploited to speed-up the detailed placement stage during global swap and Tile-level legalization stages using multi-core machines.

Chapter 4

Our Placement Flow

This chapter describes the overall placement flow. The complete placement flow is depicted in Figure 4.1. We outline the major steps here while the highlights of each step are discussed in next sections. In the first step, global solution is obtained by running a high quality row-based placer. In the next step, tile-level refinement is done to satisfy clock constraints and density constraints for LCELLS, DFF, REGFILE and BRAM. This is followed by intra-tile clock assignment to different DFF columns and legalization. Next, wirelength reduction techniques are used while maintaining the site-legality of the solution. The key highlights of the above stages are presented below.

4.1 Global Placement

The physical space available for placement of each type of cell is very discrete in our placement problem. To overcome this challenge, virtual platform and corresponding virtual netlist is generated containing only space reserved for LCELLs. A standard row-based placer is then used to generate placement solution for this platform. The result of global placer is then mapped back to original platform as explained in [7].

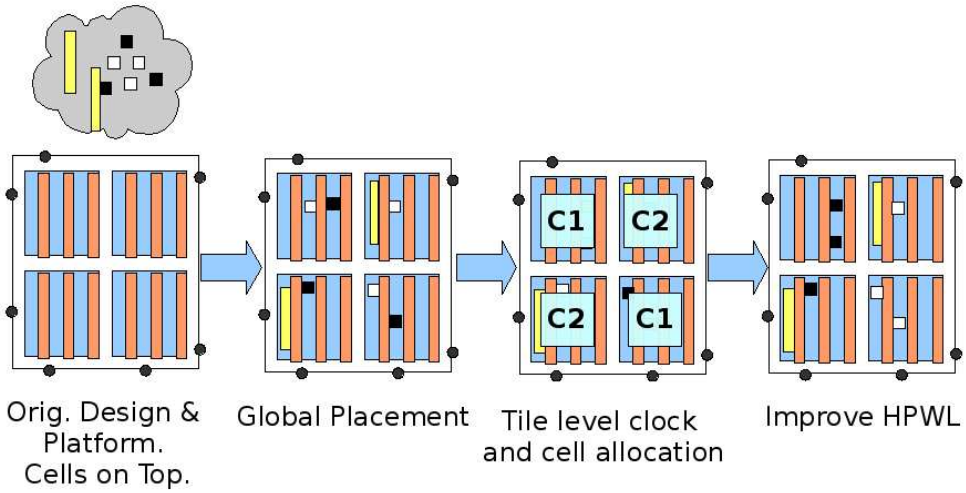

Figure 4.1: Flow of RegPlace. Different colored columns on platform represent different type compatible sites.

4.2 Global Refinement

In normal ASIC placement flow, enforcing a density of less than one would have ensured that the density of each tile is less than one. Hence, the placement would have been legalizable at tile level. However, since in case of structured ASIC each tile has different types of cells to be placed, density for each of the cells - LCELL, DFF, REG file and BRAM needs to be less than one. Moreover, clock constraints create additional density requirements for DFFs. For each tile, to ensure that the cells present inside it can be legalized, the following constraints need to be satisfied: a) The density of each type of cell should be less than one, b) Clock constraints must be satisfied e.g. the number of clocks in each flip-flop column must be one and the total number of

clocks present in each tile should be less than one, c) Even if these constraints are satisfied, the tile may be unlegalizable. For example, consider a case where 3 DFFs with clock-1 is present and one DFF with clock-2 is to be placed. The given tile has two columns of size two. Hence, density and clock constraints are satisfied but it still can't be legalized.

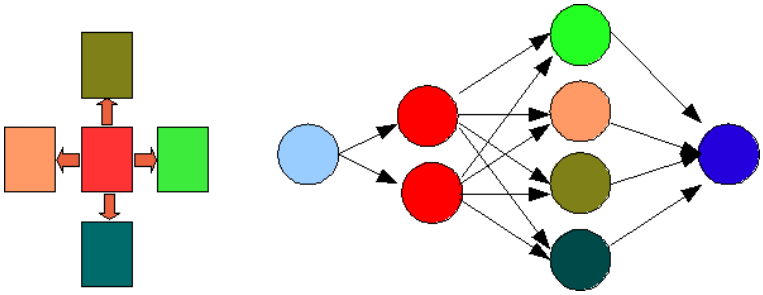

Figure 4.2: Network flow based density equalization

The first step for satisfying DFF density and clock constraint is to determine which set of clocks will be present in each tile under the constraint that no more than N different clocks exist per tile (N=4 in our implementation). Clock assignment to each tile is done using an integer linear program [7].

Once the clock assignment is fixed, we formulate the problem of determining how many DFFs to move among various tiles as a min-cost network flow problem with capacity constraints. The network flow is constructed by connecting a tile congested with respect to DFFs with the four tiles around it as shown in Figure 4.2. The maximum capacity of any edge is taken as the minimum of the number of DFFs that need to exit a congested tile and the number of DFFs which can move into neighboring tile without congesting it.

In our implementation, the cost of moving a cell out of its current tile is 1. Thus the network flow solution minimizes the *number of DFF* that need to move for satisfying density constraints. It is possible to extend our work by modeling the cost as change in wirelength due to moving a DFF which corresponds to a solving density constraints with least movement. At this stage, each tile satisfies density constraints with respect to DFFs.

4.3 Site Legalization inside Tiles

Once all of the above steps are run, each tile's density for each type of cell is under control i.e. it is possible to find a legal position for each cell. BRAMs and REGs are first placed at their best legal position by solving an assignment problem which maps them to legal positions [7]. However, the LCELLs and DFFs need to be put in their respective columns (see Figure 2.1). LCELLs are now placed by finding the nearest legal location. DFFs are legalized in two levels - first by assigning a clock-type to each column and then by moving the cells to actual location.

In the first step for tile level site legalization, we determine which *column* should be reserved for which clock (note that each column can have cells of only one type of clock). Our experiments show that this step is very critical since if the column reservation is not done smartly, wirelength can degrade significantly when moving the DFFs to column which can accommodate its clock type. We solve this problem by formulating it as a integer linear program. Each column c is assigned as many binary variables as their are clock

types. The cost function is the sum of efforts required to clean up a column by removing the DFFs whose clock type is not the same as that returned by integer linear program(ILP) solution. A column which initially does not have any DFF in it is assigned a clock number -1 indicating that it can be reserved at a later time while wirelength minimization. Consider a case where up to M DFFs can be accommodated in a column and let each tile have P DFF columns and currently C clocks in it with N_i (i $\in$ [1, C]) DFFs for each of these clock types. Binary variable p_i, when true, expresses that column p is reserved for clock i. Let the number of cells before DFF site legalization in the column p of clock type i be given by n_{pi}.

The complete ILP can be written as:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{p=1}^{P} \left(\sum_{i=1}^{C} p_i \times \left(\sum_{j=1, j\neq i}^{C} n_{pj} \right) \right) \\
\text{Subject To :} \quad & \sum_{i=1}^{C} p_i \ \leq \ 1 \qquad\qquad \forall p \in [1, P] \\
& \sum_{p=1}^{P} p_i \ \geq \ \lceil n_{pi}/M \rceil \quad \forall i \in [1, C] \\
& p_i \quad \in \ 0, 1
\end{aligned}$$

The ILP above has O(PC) binary variables and O(P + C) constraints. The cost function is the number of DFFs that need to be moved on assigning clocks to the columns. The first set of constraints make sure only one clock can occupy a column. The second set of constraints guarantee enough columns reserved within the tile for each clock type so that its cells can be placed. The values of C and P is determined by the structured ASIC's architecture and is

generally a small constant number (e.g. $C = 4$ and $P = 24$ in our benchmarks). This ILP thus assigns clocks to columns such that minimum number of DFFs need to be moved from their location.

With the assignment of all columns to a clock type or unreserved type, we iterate over the LCELLs and DFFs to legalize them. These cells are sorted in non-decreasing horizontal coordinate and are assigned to their closest unoccupied and clock compatible (in case of DFFs) possible location. A procedure *rotateAndPlace* has been implemented which takes as input a given point and a cell and rotates in circle with increasing radius around the point until the given cell can be placed legally depending on the type of the cell. Clock compatibility is maintained to while placing them for DFFs.

4.4 Wirelength Recovery

The chip and tile level density legalization outlined in the Section 4.2 and Section 4.3 produce placement results which are completely legal with respect to clock, overlap and site constraints. However, due to movement of several cells away from the initial relative order suggested by the global placer results on virtual platform, significant increase in the wirelength occurs. Our wirelength optimization procedure depicted in Algorithm 4.4 recovers this increase in wirelength. Our philosophy during wirelength minimization is to never break the clock, site or overlap legality. Algorithm 4.4 shows the process of wirelength recovery. In the first phase, large distance inter-tile movement of cells takes place. Using the median idea [12], the best bounding box (BB)

of each cell is computed. The cells are then sorted in the non-increasing order of their distance from their best BB and processed in that order. For the cell being currently optimized, a procedure returns the list of all the tiles which have non-empty geometrical intersection with its best BB. This list is iterated while trying to place the cell near the center of the intersection of best BB and the tile currently being tried using routine *rotateAndPlace*. After this stage, all inter-tile movements have been done and we focus on intra tile wirelength minimization.

Algorithm 1 Wirelength Minimization

Ensure: Placement Is Legal
{ **Large Distance Movement** of Cells}
while WL Improvement $\geq \delta$ **do**
  Compute Best BB b_c for each cell c
  Sort cells according to distance from b_c
  for all cell c in sorted list **do**
    List lst = All tiles intersecting b_c
    for all tile t in list lst **do**
      rotateAndPlace(c, center of t's intersection with b_c)
      if could place in t **then**
        if WL Improved **then**
          break
        else
          continue

4.5 Parallelization of placer

Time consumed during detailed placement is mostly dominated by wirelength recovery stage shown in section 4.4. During this stage, the best-bounding box for each cell. Then the cell is moved to its best-bounding box

if there is any empty location present. If there is no empty location present inside the best-bounding box, a location around the best-bounding box is searched which is empty. If wirelength improves by putting it at this location, it is moved there otherwise it is not moved. Parallelizing the algorithm is tricky since although movement of cells can be done independently of each other, they may be interdependence since movement of one cell may change the best bounding box of other. To handle this, a global hash table of nets is maintained for every net which was connected to a cell that is moved. Before moving any cell, the hash table is checked for all nets connected to the cell. If a net connected to the cell is found in the hash table, its bounding box is recomputed.

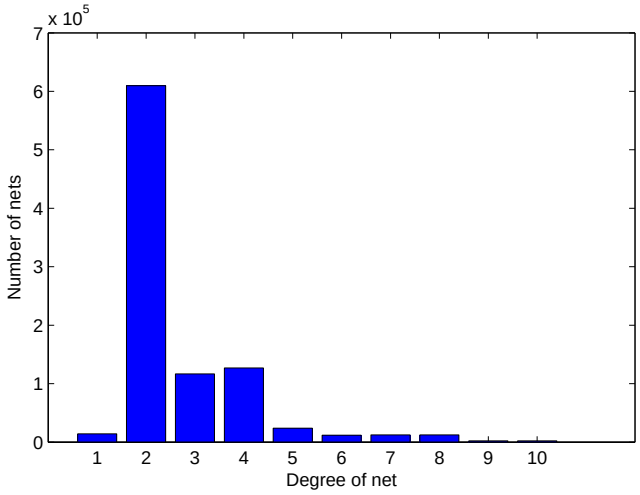

Figure 4.3: Histogram showing the number of nets vs. the number of connections for easic2 [21].

The complete algorithm for global swap in parallel is shown in Algo-

rithm 4.5. Static load balance is achieved while dividing cells to be used by different threads. DivideCellsForThreads (in Algorithm 4.5 is used for dividing the task of cell allocation for each thread. Dividing the cells on threads in a random order can be very costly if the number of interconnections between two group of cells is very high. If the interconnections are very high, bounding box needs to be computed for large number of cells making the operations slow. In general, the connectivity of cells in a design is low as shown in Figure 4.3. To avoid this, partitioning is done to minimize the number of interconnects between two group of cells. This helps in static load balancing.

Algorithm 2 Parallel Wirelength Minimization

Ensure: Placement Is Legal
{ **Large Distance Movement** of Cells}
while WL Improvement $\geq \delta$ **do**
  Compute Best BB b_c for each cell c
  Initialize Hash Table h
  DivideCellsForThreads(all c, numberOfThreads)
  for all thread **do**
    Sort cells according to distance from b_c
    for all cell c in sorted list **do**
      List lst = All tiles intersecting b_c
      for all tile t in list lst **do**
        Update bb using hash
        rotateAndPlace(c, center of t's intersection with b_c)
        if could place in t **then**
          if WL Improved **then**
            break
          else
            continue

Algorithms 4.5 and 4.5 describe the whole process. To avoid overwriting two cells on same location by different threads, mutual exclusion is used as

Algorithm 3 rotateAndPlace

```
1: for all cell c do
2:     Find placable location
3:     Mutex Lock
4:     place(c,p)
5:     update(h,c)
6:     Mutex Unlock
7:     return
```

shown in Algorithm 4.5. As shown in line 4 of Algorithm 4.5, task allocation for each thread is done by dividing cells using DivideCellsForThreads. Once cells are divided, they are sent on different threads to be placed. Each thread tries to place the cells in a manner similiar to Algorithm 4.4 while keeping the hash table updated.

Chapter 5

Experimental results

All the above algorithms are implemented in C++ in our placer tool RegPlace (REGular PLACEr). Experiments were conducted on two machines - one with two cores 3.3 GHz 64 bit AMD linux with 4GB RAM and the other with eight cores 2.4GHz 64 bit Intel Xeon with 4GB RAM. We used GLPK [2] as the ILP and network flow solver. pThread was used for parallel implementations of algorithms.

Table 5.1 shows the characteristics of the benchmarks. The benchmarks were provided by eASIC [1] as part of the placement challenge. mPL [9] and CAPO [20] were used to generate the global placement solution. Table 5.2 shows the characteristics of the platform on which they are to be placed. Platform B has bigger capacity and can place designs with higher number of LCELLs, DFFs, BRAMs and REGs. For the benchmarks shown in Table 5.1, only easic3 is placed on platform B. Figure 5.1 shows a comparison of different benchmarks. The benchmarks cover different spectrums of complexity. easic4 is a small benchmark with around 1/10 of the total number of cells from the bigger benchmarks. easic3 is the biggest benchmark. easic1 and easic5 have the highest number of clocks, hence clock constraints have biggest impact on

them.

Table 5.1: Benchmark Characteristics

Bench	#LCELL	#DFF	#BRAM	#REG	#CLK
easic1	832,824	87,052	110	172	18
easic2	812,200	45,478	175	686	7
easic3	961,063	52,780	192	0	3
easic4	102,038	23,330	0	44	7
easic5	913,853	84,505	145	262	26

Table 5.2: Platform Characteristics

Plat.	Max LCELL	Max DFF	Max REG	Max BRAM	Intertile Space	
					Horz	Vert
A	1M	675k	1760	440	8.00	6.45
B	1.2M	811k	2112	528	8.00	8.92

First, we analyze the impact of ILP based Tile legalization shown in section 4.3. For comparison, a heuristic column assignment was implemented. First the minimum number of columns required for each clock type was calculated inside a tile. Clocks were assigned to each column in two loops over all columns. In first iteration, a clock was assigned to a column if the number of DFFs of that clock was maximum in that column, while making sure the number of columns assigned to a clock don't exceed the minimum allowed for that clock. In second iteration, clock assignment was done while giving priority to clocks which were less than the minimum number of columns. Finally, all unassigned columns were assigned a clock based on the clock driving the maximum number of DFFs in that column. This algorithm ensures that all DFFs will be legalizable while making sure minimum number of DFFs need to be moved. Increase in HPWL is shown due to this method vs our method

during Tile legalization in Table 5.3. As can be seen, our method shows an average 3.7X improvement over the heuristic method of column assignment.

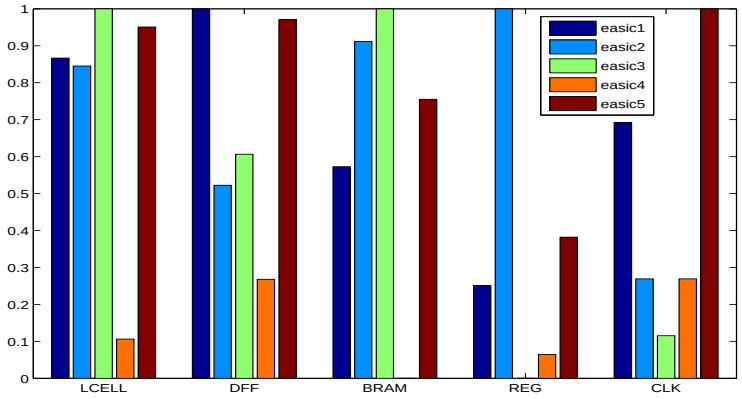

Figure 5.1: Bargraph showing the relative distribution of LCELL, DFF, REG, BRAM and clock for different benchmarks.

Table 5.3: Increase in HPWL during tile-level site legalization

Benchmark	Heuristic column assignment	Our Algorithm	Improvement
easic1	467k	107k	4.3X
easic2	1306k	341k	3.8X
easic3	2060k	373k	5.5X
easic4	380k	56k	6.8X
easic5	2290k	883k	2.6X

Next, we show the wirelength at different stages of placement during our flow in Table 5.4. As can be seen, due to clock and regularity constraints, there is high increase in wirelength during tile-level legalization. A lot of this increase in wirelength is recovered during the next stage.

Table 5.4: Wirelength after different stages of placement

	easic1	easic2	easic3	easic4	easic5
After Global Placement	14.0M	25.2M	20.2M	2.1M	16.7M
After Global Refinement	15.8M	25.2M	20.2M	2.1M	16.7M
After Tile-level legalization	18.1M	26.8M	21.9M	2.4M	19.3M
After Wirelength Recovery	14.1M	25.4M	20.4M	2.1M	16.8M

5.1 Overall Wirelength

A preliminary version of RegPlacer [7] competed and won the eASIC placement contest [1]. In its current form, we have further improved the wirelength of RegPlacer by 9%. Table 5.5 shows the comparative data for our placer vs. other participants Team1 and Team2 [5]. The wirelength (WL) numbers for Team1, Team2 are as reported in the contest results. To understand the impact of using different global placers to generate an initial solution on the virtual platform, we repeated our placer flow with two state-of-the-art placer: mPL and CAPO. The initials (M, C) in last four column names depict use of the above placers respectively.

Table 5.5: Comparison of our placer with other contestants.

Bench	Team1	Team2	RegPlacer[7] (M)	RegPlacer[7] (C)
	WL	WL	WL	WL
easic1	16.9M	10.6M	11.3M	12.8M
easic2	32.3M	25.8M	23.4M	25.1M
easic3	20.9M	20.5M	18.8M	19.6M
easic4	2.3M	1.8M	2.0M	2.1M
easic5	20.4M	14.4M	13.9M	16.6M
Total	92.8M	73.1M	69.4M	76.2

From Table 5.5, several observations can be made. Our wirelength

results improve significantly (10%) when the initial placement on virtual platform is generated using mPL rather than CAPO. Compared to other teams, RegPlacer beats Team1 in all the benchmarks irrespective of the initial placer solution being generated with CAPO or mPL. The wirelength reduction with respect to Team1 is as much as 33%. RegPlacer when used with mPL for initial placement, is better than Team2 in 3 out of 5 benchmarks as well as the total wirelength. The reduction in total wirelength compared to Team2 is 5%. These results show that RegPlace is a high quality solution for structured ASIC placement problem. Next, we conducted experiments to show the effect of parallelization of detailed placer on two-core and eight-core machines.

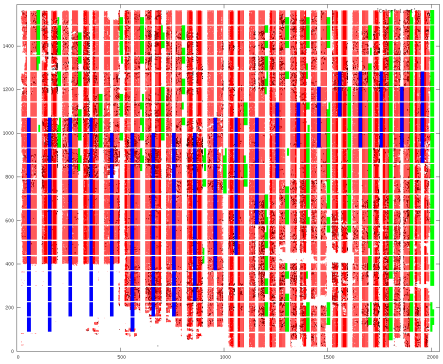

Figure 5.2: Final layout of benchmark easic2. Cells are shown as LCELL(red), DFF(black), REG(green), and BRAM(blue).

Figures 5.3 and 5.4 show the speedup achieved during wirelength recovery stage on two-core and eight-core machines. On both these machines, experiments were conducted by varying the number of threads spawned. On a two-core machine, effect of parallelization saturates on going beyond more

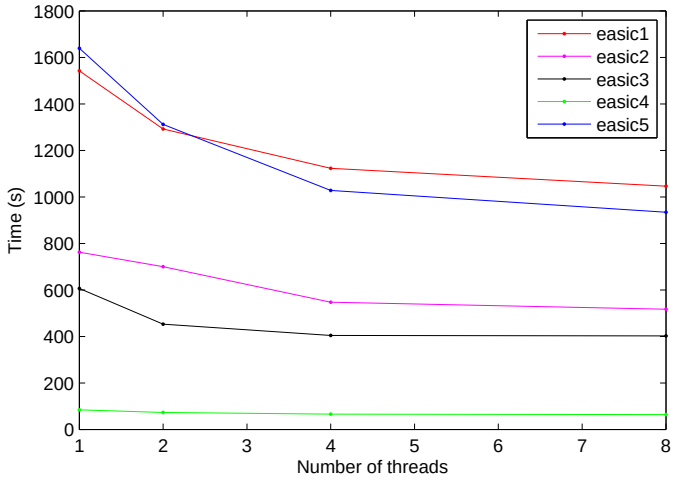

Figure 5.3: Speedup for different benchmark with number of threads on two-core machine.

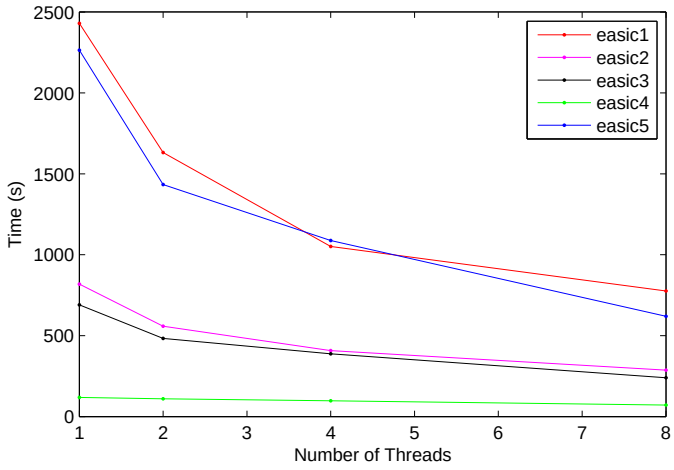

Figure 5.4: Speedup for different benchmark with number of threads on eight-core machine.

than two threads quickly. On an eight core machine, parallelization gives a significant performance improvement for eight threads. Up to 3x improvement is achieved using parallelization. Figure 5.2 shows the output of our placer for illustration for the benchmark easic2 where each cell type has been plotted with a different color.

Chapter 6

Conclusions

In this work, we proposed a new flow for efficient solution of placement problem for structure ASICs. The key contributions of this work are: a) a detail placement flow specifically for structured ASIC, b) ILP based clock constraint to assign DFFs to columns , c) parallelizing the detailed placement stage to get high speed-up. Powered by these techniques, our placer won the eASIC placement challenge. Experimental results show that we obtain as much as 33% less wirelength than the competing teams and up to a 3X speedup by parallelizing.

Bibliography

[1] eASIC Corporate Website. `http://www.easic.com/`.

[2] GNU Linear Programming Kit. `http://www.gnu.org/software/glpk/`.

[3] SOC Central. `http://www.soccentral.com/`.

[4] SOC Encounter tool. `http://www.cadence.com/products/di/soc_encounter`.

[5] The Esteem Placer, written by Bob Erickson, an independent software consultant. `http://www.linkedin.com/in/boberickson`.

[6] Vaughn Betz and Jonathan Rose. Vpr: A New Packing, Placement and Routing Tool for FPGA Research. In *FPL '97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pages 213–222, London, UK, 1997. Springer-Verlag.

[7] Ashutosh Chakraborty, Anurag Kumar, and David Z. Pan. Regplace: A High Quality Open-source Placement Framework For Structured ASICs. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 442–447, 2009.

[8] Pak K. Chan and Martine D. F. Schlag. Parallel placement for field-programmable gate arrays. In *FPGA '03: Proceedings of the 2003*

ACM/SIGDA eleventh international symposium on Field programmable gate arrays, pages 43–50, New York, NY, USA, 2003. ACM.

[9] Tony F. Chan, Jason Cong, Joseph R Shinnerl, Kenton Sze, and Min Xie. mpl6: Enhanced Multilevel Mixed-size Placement. In *ISPD '06: Proceedings of the 2006 international symposium on Physical design*, pages 212–214, New York, NY, USA, 2006. ACM.

[10] J. Cong and Y. Zou. Parallel Multi-level Analytical Global Placement on Graphics Processing Units. *Computer-Aided Design, International Conference on*, 0, 2009.

[11] Ken Eguro, Scott Hauck, and Akshay Sharma. Architecture-adaptive Range Limit Windowing for Simulated Annealing FPGA Placement. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 439–444, New York, NY, USA, 2005. ACM.

[12] S. Goto. An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit layout. *Circuits and Systems, IEEE Transactions on*, 28(1):12–18, Jan 1981.

[13] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[14] Hao Li and Yue Zhuo. Criticality History Guided FPGA Placement Algorithm for Timing Optimization. In *GLSVLSI '08: Proceedings of*

the 18th ACM Great Lakes symposium on VLSI, pages 267–272, New York, NY, USA, 2008. ACM.

[15] Andrew C. Ling, Deshanand P Singh, and Stephen D. Brown. Incremental Placement for Structured ASICs Using the Transportation Problem. *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, pages 172–177, Oct. 2007.

[16] Tao Luo and David Z. Pan. DPlace2.0: A Stable and Efficient Analytical Placement Based on Diffusion. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 346–351, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[17] Takumi Okamoto, Tsutomu Kimoto, and Naotaka Maeda. Design Methodology and Tools for NEC Electronics' Structured ASIC ISSP. In *ISPD '04: Proceedings of the 2004 international symposium on Physical design*, pages 90–96, New York, NY, USA, 2004. ACM.

[18] Min Pan, N. Viswanathan, and C. Chu. An Efficient and Effective Detailed Placement Algorithm. *Computer-Aided Design, International Conference on*, 0:48–55, 2005.

[19] C. C. Pettey and M. R. Leuze. Parallel placement of parallel processes. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 232–238, New York, NY, USA, 1988. ACM.

[20] Jarrod A. Roy, David A. Papa, Saurabh N. Adya, Hayward H. Chan, Aaron N. Ng, James F. Lu, and Igor L. Markov. Capo: Robust and Scalable Open-source min-cut Floorplacer. In *ISPD '05: Proceedings of the 2005 international symposium on Physical design*, pages 224–226, New York, NY, USA, 2005. ACM.

[21] Herman Schmit, Amit Gupta, and Radu Ciobanu. Placement Challenges for Structured ASICs. In *ISPD '08: Proceedings of the 2008 international symposium on Physical design*, pages 84–86, New York, NY, USA, 2008. ACM.

[22] N. Viswanathan, Min Pan, and C. Chu. FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion control. In *ASP-DAC '07: Proceedings of the 2007 conference on Asia South Pacific design automation*, pages 135–140, Washington, DC, USA, 2007. IEEE Computer Society.

[23] Yue Zhuo, Hao Li, Qiang Zhou, Yici Cai, and Xianlong Hong. New Timing and Routability Driven Placement Algorithms for FPGA Synthesis. In *GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pages 570–575, New York, NY, USA, 2007. ACM.

Vita

Anurag Kumar was born in India on 6 March, 1983. He did his schooling in northern parts of India. He received his undergraduate degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur. After his undergrad, he worked as an engineer in Atrenta India Pvt. Ltd., an Electronic Design Automation Company. He has also interned at National University of Singapore, Intel and Intrinsity. In Fall 2007, he enrolled in the masters program in the Department of Electrical and Computer Engineering at the Univeristy of Texas at Austin.

Permanent address: 3106 Duval St, Apt. 210, Austin, Tx 78705

This report was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.